

EXHIBIT E

```
0 /* Copyright Avenail Corporation 1997-2000; All Rights Reserved */
1 /* sslenv.c is the SSL environment file; it defines functions used by
2    the SSL module as callbacks for managing mutexes, memory, I/O,
3    and user interaction. */
4
5 #include "sslmain.h"
6 #include "ssldap.h"
7 #include "ldapcert.h"
8 #include <global.h>
9 #include <bsafe.h>
10 #include <pkcs.h>
```

<due to the size of this file and the small portion of it which is relevant here, we include only the single function SSLEncode>

```
970 int FAR EXPORT SSLEncode(S5Packet *ibuf, S5Packet *obuf, int flag, void *handle)
971 {
972     S5SSLHandle *ref = handle;
973     S5SSLFlowConnection *conn = &(ref->conn);
974     SSLContext *ctx = ref->ctx;
975     uint32 ilen;
976     uint32 len = ibuf ? ibuf->len : 0;
977     int sloppy = 0;
978
979 #ifdef HYPER_DEBUG
980     int i;
981 #endif
982     SSLErr err;
983     uint32 wrtp = 0;
984
985     if (flag & S5_STATE_DUMP)
986     {
987         SSLBuffer block;
988         unsigned totalSize;
989         PSSLStateDump dump = (PSSLStateDump)obuf->data;
990
991         if (obuf->len < 4096)
992         {
993             obuf->len = 4096;
994             return ENCODE_BUFFER_TOO_SMALL;
995         }
996
997         // get the SSL state
998         //
999         if ((err = SSLEXPCTX(ctx, &block)) != SSLNoErr)
1000         {
1001             if (GlobalUpdate)
1002                 GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_ERROR,
1003                             IDS_SSL_EXPORTCONTEXTFAILED, err);
1004             return -1;
1005         }
1006
1007         // compute the total size of the data
1008         //
1009         totalSize = block.length + sizeof(SSLStateDump);
1010
1011         // validate the output buffer size
1012         //
1013         if (obuf->len < totalSize)
1014         {
1015             obuf->len = totalSize;
1016             SSLFreeBuffer(&block, &ctx->sysCtx);
1017             return ENCODE_BUFFER_TOO_SMALL;
1018         }
1019
1020         // put the SSLStateDump structure at the beginning of the output buffer
1021         //
1022         dump->SSLContext = ctx;
1023         dump->ContextSize = sizeof(SSLContext);
1024         dump->SSLState.data = (uint8 *) (dump+1);
```

```
1025         dump->SSLState.length = block.length;
1026
1027         // copy the SSL state to the output buffer
1028         //
1029         memcpy(dump->SSLState.data, block.data, block.length);
1030         obuf->len = totalSize;
1031         SSLFreeBuffer(&block, &ctx->sysCtx);
1032
1033         if (GlobalUpdate)
1034             GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1035                          IDS_SSL_EXPORTEDCONTEXT, obuf->len);
1036
1037         return 0;
1038     }
1039
1040     if(ref->endtime > 0)
1041         if(time((time_t *) NULL) >= ref->endtime) {
1042             if(GlobalUpdate)
1043                 GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1044                              IDS_SSL_LIFETIMEEXCEEDED);
1045
1046             return -1;
1047         }
1048
1049     SSLGetWritePendingSize(ctx, &wrtp);
1050 #ifdef HYPER_DEBUG
1051     if(wrtp)
1052         if(GlobalUpdate)
1053             GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1054                          IDS_SSL_BYTESPENDINGWRITE, wrtp);
1055 #endif
1056
1057     if(flag & S5_DATAGRAM)
1058         if(ref->ssloppy)
1059             {
1060                 if((rt = SSLSetSloppyMode(ctx, 1)) != SSLNoErr)
1061                     {
1062                         if(GlobalUpdate)
1063                             GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_WARNING,
1064                                          IDS_SSL_SSLOPPYMODEFAILED, rt);
1065                         return -1;
1066                     }
1067                 ssloppy = 1;
1068             }
1069         else
1070             {
1071                 /* UDP naked, baby! */
1072                 #ifdef HYPER_DEBUG
1073                 if(GlobalUpdate)
1074                     GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1075                                  IDS_SSL_GOTDATAGRAM, flag, ibuf->len);
1076                 #endif
1077                 #ifdef OPTIMIZE_UDP_NAKED
1078                 /* this is much cleaner and faster, but causes inconsistency in the
1079                  API from the caller. Sigh. */
1080                 *obuf = *ibuf;
1081                 #else
1082                 #ifdef WIN32
1083                 obuf->data = HeapAlloc(GetProcessHeap(), 0, ibuf->len);
1084                 #else
1085                 obuf->data = malloc(ibuf->len);
1086                 #endif
1087                 if(obuf->data == NULL) {
1088                     #ifdef HYPER_DEBUG
1089                     FPRINTF(stderr, _T("Returning error, buf data is null in
1090                                obufdata\n"));
1091                     #endif
1092                     return SSLMemoryErr;
1093                 }
1094                 memcpy(obuf->data, ibuf->data, ibuf->len);
1095             }
```

```
1094             obuf->len = ibuf->len;
1095 #endif
1096             return ibuf->len;
1097         }
1098
1099
1100
1101     if(flag & S5_ENCODE) {
1102
1103 #ifdef HYPER_DEBUG
1104         if(GlobalUpdate)
1105             GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1106                           IDS_SSL_ENCODINGBYTES, ibuf->len);
1107 #ifndef AUTOSOCKS
1108         for(i = 0; i < ibuf->len; i++)
1109             FPRINTF(stderr, _T("%02x "), ibuf->data[i]);
1110             FPRINTF(stderr, _T("\n"));
1111 #endif
1112 #endif
1113
1114 #define SSL_MAX_ENCODE_SIZE 4096
1115
1116 #ifdef SSL_MAX_ENCODE_SIZE
1117         if(ibuf->len > SSL_MAX_ENCODE_SIZE) {
1118             conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1119                                     IDS_SSL_MAXENCODESIZEEXCEEDED,
1120                                     ibuf->len, SSL_MAX_ENCODE_SIZE);
1121             ibuf->len = SSL_MAX_ENCODE_SIZE;
1122         }
1123 #endif
1124
1125         if(obuf->data != NULL) {
1126             if(obuf->len < (int) (ibuf->len + SSL_HEADLEN + 64 + wrtp)) {
1127                 conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_DEBUG,
1128                                         IDS_SSL_BUFFERTOOSHORT,
1129                                         obuf->len,
1130                                         + 64 + wrtp));
1131                 obuf->len = (int) ibuf->len + SSL_HEADLEN + 64 + wrtp;
1132                 if(ssloppy) SSLSetSloppyMode(ctx, 0);
1133                 return ENCODE_BUFFER_TOO_SMALL;
1134             }
1135             conn->writebuffer.data = obuf->data;
1136             conn->writebuffer.len = obuf->len;
1137             conn->writebuffer.off = SSL_HEADLEN;
1138             conn->writeflag = SSL_FLOW_WRITE_NOMAKEBUF;
1139         } else
1140             conn->writeflag = 0;
1141
1142         ilen = (uint32) ibuf->len;
1143         if((err = SSLWrite(ibuf->data, &ilen, ctx))) {
1144             conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_ERROR,
1145                                     IDS_SSL_WRITEERROR, err);
1146             if(ssloppy) SSLSetSloppyMode(ctx, 0);
1147             return -1;
1148         }
1149
1150         if(conn->writebuffer.off > 0xFFFF) {
1151             conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_ERROR,
1152                                     IDS_SSL_PACKETTOOBIG,
1153                                     conn->writebuffer.off);
1154             if(ssloppy) SSLSetSloppyMode(ctx, 0);
1155             return -1;
1156         }
1157
1158         if(obuf->data != NULL) {
1159             /* Here we shift the semantics of writebuffer; off now points
1160              to the beginning of the data, and len points to the end of
1161              the data, not the length of the buffer, which we no longer
1162              need to know since we don't be depositing anything new in it */
1163             obuf->len = conn->writebuffer.off;
```

```
1163         conn->writebuffer.len = conn->writebuffer.off;
1164         conn->writebuffer.off = SSL_HEADLEN;
1165     } else {
1166         if(conn->writebuffer.off == SSL_HEADLEN)
1167             /* Wow! Some thoughtful soul in SSLFlowWrite has left us a 4
1168              byte offset in the writebuffer so we can insert our header
1169              without needing to malloc a new buffer and memcpy into it! */
1170             obuf->data = conn->writebuffer.data;
1171         else {
1172             #ifndef _WINDOWS
1173                 obuf->data = malloc(conn->writebuffer.len + SSL_HEADLEN);
1174             #else
1175             #ifdef WIN32
1176                 obuf->data = HeapAlloc(GetProcessHeap(), 0,
1177                                     conn->writebuffer.len +
1178                                     SSL_HEADLEN);
1179             #endif
1180             if(obuf->data == NULL) {
1181                 conn->modctx.log.update(sslLogHandle, S5_LOG_MISC,
1182                                     S5_LOG_ERROR, IDS_SSL_MALLOCFAILED);
1183                 if(ssloppy) SSLSetSloppyMode(ctx, 0);
1184                 return -1;
1185             }
1186             memcpy(obuf->data + SSL_HEADLEN,
1187                   conn->writebuffer.data + conn->writebuffer.off,
1188                   conn->writebuffer.len - conn->writebuffer.off);
1189             #ifdef WIN32
1190                 HeapFree(GetProcessHeap(), 0, conn->writebuffer.data);
1191             #else
1192                 free(conn->writebuffer.data);
1193             #endif
1194         }
1195     }
1196     obuf->len = (int) (conn->writebuffer.len -
1197                     conn->writebuffer.off + SSL_HEADLEN);
1198 }
1199
1200 obuf->data[0] = SSL_HEADVERSION;
1201 obuf->data[1] = conn->state;
1202 obuf->data[2] = (uint8) ((conn->writebuffer.len - conn->writebuffer.off)
1203                        >> 8);
1204 obuf->data[3] = (uint8) ((conn->writebuffer.len - conn->writebuffer.off)
1205                        & 0xFF);
1206
1207 conn->writebuffer.data = NULL;
1208 conn->writebuffer.len = 0;
1209 conn->writebuffer.off = 0;
1210 if(GlobalUpdate)
1211     GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1212                 IDS_SSL_ENCODERRETURNING, obuf->len, ilen);
1213 #ifdef HYPER_DEBUG
1214 #ifndef AUTOSOCKS
1215     for(i = 0; i < obuf->len; i++)
1216         FPRINTF(stderr, _T("%02x "), obuf->data[i]);
1217     FPRINTF(stderr, _T("\n"));
1218 #endif
1219 #endif
1220 if(ssloppy) SSLSetSloppyMode(ctx, 0);
1221 return (int) ilen;
1222 }
1223
1224 /* we must be decoding, instead of encoding.. */
1225 #ifdef HYPER_DEBUG
1226 if(GlobalUpdate)
1227     GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1228                 IDS_SSL_DECODINGBYTES, ibuf->len);
1229 if(GlobalUpdate)
1230     GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1231                 IDS_SSL_READBUFFERCOMINGIN,
```

```
1232             conn->readbuffer.len - conn->readbuffer.off);
1233 #ifndef AUTOSOCKS
1234     for(i = 0; i < ibuf->len; i++)
1235         FPRINTF(stderr, _T("%02x "), ibuf->data[i]);
1236     FPRINTF(stderr, _T("\n"));
1237 #endif
1238 #endif
1239
1240     if(ibuf->len < SSL_HEADLEN)
1241     {
1242         conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_ERROR,
1243                                IDS_SSL_DECODEINCOMPLETEPACKET);
1244         if(ssloppy) SSLSetSloppyMode(ctx, 0);
1245         return -1;
1246     }
1247
1248     if(ibuf->data[0] != SSL_HEADVERSION) {
1249         conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_ERROR,
1250                                IDS_SSL_HEADERVERSIONMISMATCH,
1251                                SSL_HEADVERSION, ibuf->data[0]);
1252         if(ssloppy) SSLSetSloppyMode(ctx, 0);
1253         return -1;
1254     }
1255     if(ibuf->data[1] != conn->state) {
1256         conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_ERROR,
1257                                IDS_SSL_HEADERSTATEMISMATCH,
1258                                conn->state, ibuf->data[1]);
1259         if(ssloppy) SSLSetSloppyMode(ctx, 0);
1260         return -1;
1261     }
1262
1263     ilen = ((uint8) ibuf->data[2]) << 8;
1264     ilen |= (uint8) ibuf->data[3];
1265     ilen += SSL_HEADLEN; /* we must include the header in the length because
1266                          no man is an ilen. Er, because it's the length
1267                          of the whole record, including the header. */
1268
1269 #ifdef HYPER_DEBUG
1270     if(GlobalUpdate)
1271         GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1272                     IDS_SSL_PACKETSIZE, ilen);
1273 #endif
1274
1275     if(ibuf->len < (int) (ilen)) {
1276         conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_ERROR,
1277                                IDS_SSL_DECODEINCOMPLETEPACKET);
1278         if(ssloppy) SSLSetSloppyMode(ctx, 0);
1279         return -1;
1280     }
1281
1282 #if 0
1283     if(ibuf->len > (int) (ilen)) {
1284         conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_ERROR,
1285                                IDS_SSL_DECODEOVERFULLPACKET);
1286         if(ssloppy) SSLSetSloppyMode(ctx, 0);
1287         return -1;
1288     }
1289 #endif
1290
1291     SSLGetReadPendingSize(ctx, &wrtp); /* this should be zero, but seems to
1292                                         not always be, so we be safe.. */
1293
1294 #if 0
1295     /* we need to choose the size of the obuf here; since SSL adds some
1296        boundary information the size of the input should be big enough.
1297        If SSL+ starts to support compression this assumption will have
1298        to change. */
1299     len = conn->readbuffer.len - conn->readbuffer.off + wrtp + ilen;
1300 #else
1301     /* OK, so we decided to change it. Now we know the record can't
1302        be larger than 16K. */

```

```
1303 /* len = conn->readbuffer.len - conn->readbuffer.off + wrtp + 16384; */
1304 /* Hmm.. the above seems to cause telnet to studder, while a fixed 32k
1305    size fixes it; so 32k it shall be... */
1306 len = 32767;
1307 #endif
1308 if(obuf->data != NULL) {
1309     if(obuf->len < (int) len) {
1310         if(GlobalUpdate)
1311             GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_DEBUG,
1312                 IDS_SSL_BUFFERTOOSMALL, obuf->len, len);
1313         obuf->len = (int) len;
1314         if(ssloppy) SSLSetSloppyMode(ctx, 0);
1315         return ENCODE_BUFFER_TOO_SMALL;
1316     }
1317 } else {
1318 #ifndef _WINDOWS
1319     obuf->data = (unsigned char *) malloc(len);
1320 #else
1321 #ifdef WIN32
1322     obuf->data = HeapAlloc(GetProcessHeap(), 0, len);
1323 #endif
1324 #endif
1325 }
1326 #if 0
1327 /* must read all the data we can.. */
1328 len = ilen + conn->readbuffer.len;
1329 #endif
1330
1331 if (conn->readbuffer.data == NULL) {
1332 /*     conn->readbuffer.data = malloc((size_t) (len - SSL_HEADLEN)); */
1333     /* Try to re-use the input buffer instead of needing to create
1334        a new one and memcpy into it. readflag lets us know we did
1335        this so we don't try to change or free the space later */
1336     conn->readbuffer.data = ibuf->data;
1337     conn->readbuffer.len = ilen;
1338     conn->readbuffer.off = SSL_HEADLEN;
1339     conn->readflag = SSL_FLOW_READ_NOOWNBUF;
1340 } else {
1341     conn->readbuffer.data = realloc(conn->readbuffer.data,
1342         (size_t) (len - SSL_HEADLEN));
1343     conn->readflag = 0;
1344     if(conn->readbuffer.data == NULL) {
1345         conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_ERROR,
1346             IDS_SSL_REALLOCFAILED);
1347         if(ssloppy) SSLSetSloppyMode(ctx, 0);
1348         return -1;
1349     }
1350     memcpy(conn->readbuffer.data + conn->readbuffer.len,
1351         ibuf->data + SSL_HEADLEN, (size_t) (ilen - SSL_HEADLEN));
1352     conn->readbuffer.len += ilen - SSL_HEADLEN;
1353 }
1354
1355 if((err = SSLRead((void *) obuf->data, &len, ctx)) &&
1356     (err != SSLWouldBlockErr)) {
1357     conn->modctx.log.update(sslLogHandle, S5_LOG_MISC, S5_LOG_ERROR,
1358         IDS_SSL_READERROR, err);
1359     if(ssloppy) SSLSetSloppyMode(ctx, 0);
1360     return -1;
1361 }
1362 obuf->len = (int) len;
1363 #ifdef HYPER_DEBUG
1364 if(GlobalUpdate)
1365     GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1366         IDS_SSL_ENCODERRETURNINGBYTES,
1367         ilen, len);
1368 if(GlobalUpdate)
1369     GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_VERBOSE,
1370         IDS_SSL_READBUFFERGOINGOUT,
1371         conn->readbuffer.len - conn->readbuffer.off);
1372 #ifndef AUTOSOCKS
1373     for(i = 0; i < len; i++)
```

```
1374     FPRINTF(stderr, _T("%02x "), obuf->data[i]);
1375     FPRINTF(stderr, _T("\n"));
1376 #endif
1377 #endif
1378     if(conn->readflag & SSL_FLOW_READ_NOOWNBUF)
1379         if (conn->readbuffer.off < conn->readbuffer.len) {
1380             BYTE *t;
1381
1382             if(GlobalUpdate)
1383                 GlobalUpdate(sslLogHandle, S5_LOG_MISC, S5_LOG_WARNING,
1384                               IDS_SSL_ENCODELEAVINGDATA,
1385                               conn->readbuffer.len - conn->readbuffer.off);
1386             t = malloc(conn->readbuffer.len - conn->readbuffer.off);
1387             memcpy(t, conn->readbuffer.data + conn->readbuffer.off,
1388                  conn->readbuffer.len - conn->readbuffer.off);
1389             conn->readbuffer.data = t;
1390             conn->readbuffer.len = conn->readbuffer.len - conn->readbuffer.off;
1391             conn->readbuffer.off = 0;
1392         } else {
1393             conn->readbuffer.data = NULL;
1394             conn->readbuffer.off = 0;
1395             conn->readbuffer.len = 0;
1396         }
1397     if(ssloppy) SSLSetSloppyMode(ctx, 0);
1398     return (int) ilen;
1399 }
```